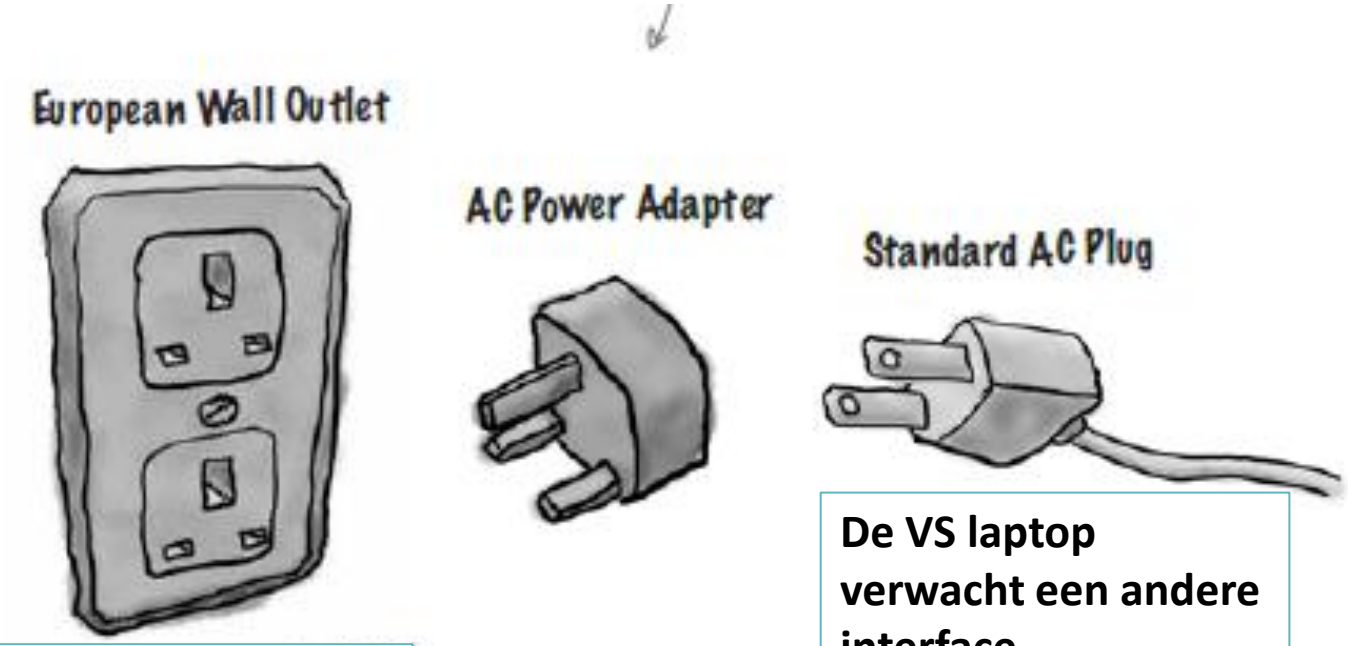
HoGent

BEDRIJF
EN
ORGANISATIE

# Adapter

## Wees adaptief

# Adapters om ons heen

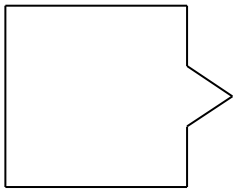



De Europese contactdoos toont een interface

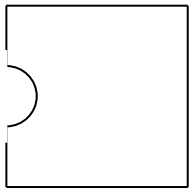
De adapter converteert de ene interface in de andere

De VS laptop verwacht een andere interface

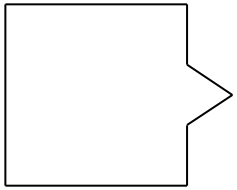
# Objectgeoriënteerde adapters



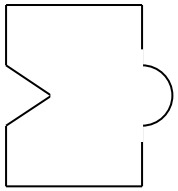
Je bestaande systeem



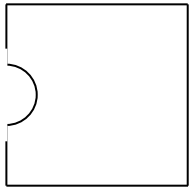
leverancierssklassen



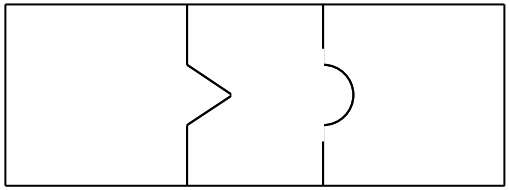
Je bestaande systeem



Adapter



leverancierssklassen



Geen veranderingen in de code

nieuwe code

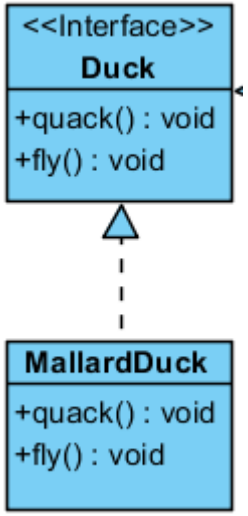
Geen veranderingen in de code

# Als het loopt en kwaakt als een eend…

- Als het loopt als een eend en kwaakt als een eend, ~~moet~~ kan het een ~~eend~~ kalkoen zijn, die in een eendenpak verpakt is ….

# Als het loopt en kwaakt als een eend…

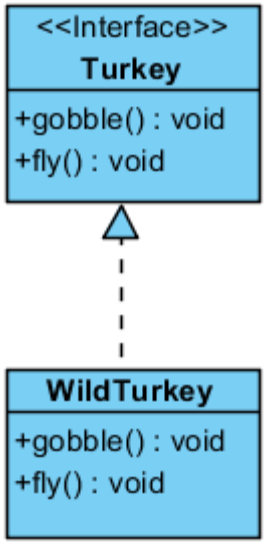
- Een vereenvoudigde eendeninterface




```
public interface Duck {

   void quack();
   void fly();

}
```

```
public class MallardDuck implements Duck {

   public void quack() {
      System.out.println("Quack");
   }

   public void fly() {
      System.out.println("I'm flying");
   }

}
```

# Tijd om kennis te maken met de nieuwe vogel

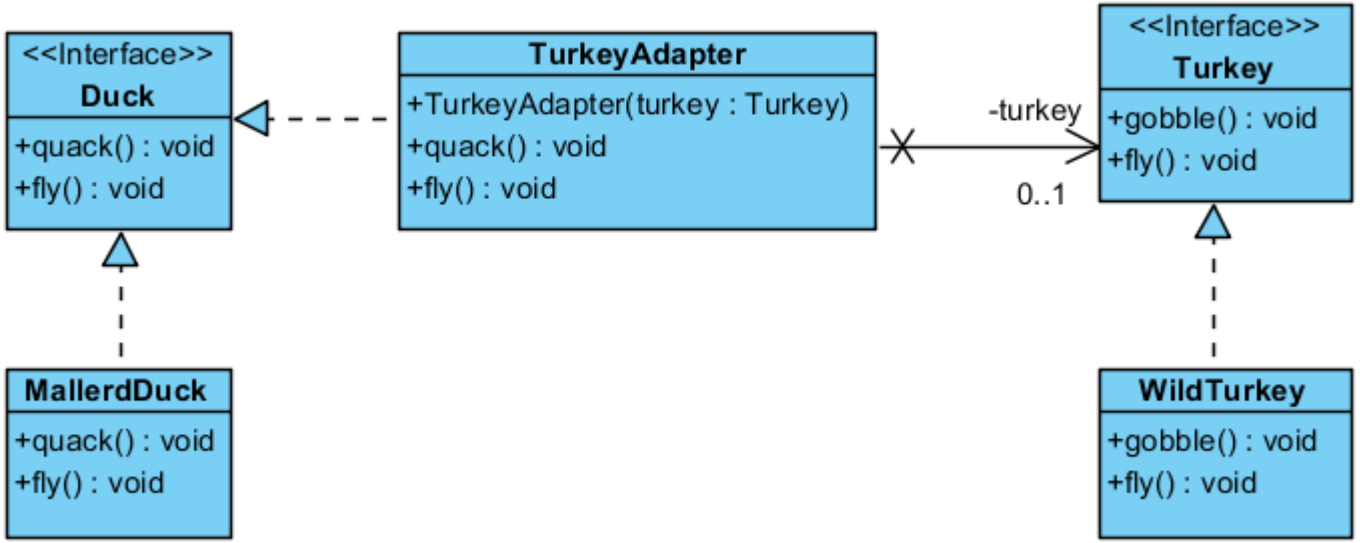



```
public interface Turkey {

    void gobble();
    void fly();

}
```

```
public class WildTurkey implements Turkey {

    public void gobble() {
        System.out.println("Gobble gobble");
    }

    public void fly() {
        System.out.println("I'm flying a short distance");
    }

}
```

# De TurkeyAdapter

- Stel je hebt een gebrek aan Duck-objecten en wilt in plaats daarvan Turkey-objecten gebruiken. We kunnen kalkoenen niet rechtstreeks gebruiken, want ze hebben een andere interface.
- Adapter schrijven



# De TurkeyAdapter

```
public class TurkeyAdapter implements Duck {

   private Turkey turkey;
   public TurkeyAdapter(Turkey turkey) {
      this.turkey = turkey;
   }

   public void quack() {
      turkey.gobble();
   }

   public void fly() {
      for (int i = 0; i < 5; i++) {
         turkey.fly();
      }
   }
```

Implementeer de nieuwe interface. Di de interface die de client verwacht

Zorg voor een referentie naar het object dat we aanpassen. Hier bvb in de constructor.

De vertaling van quack() is eenvoudig : roep gewoon de methode gobble() aan

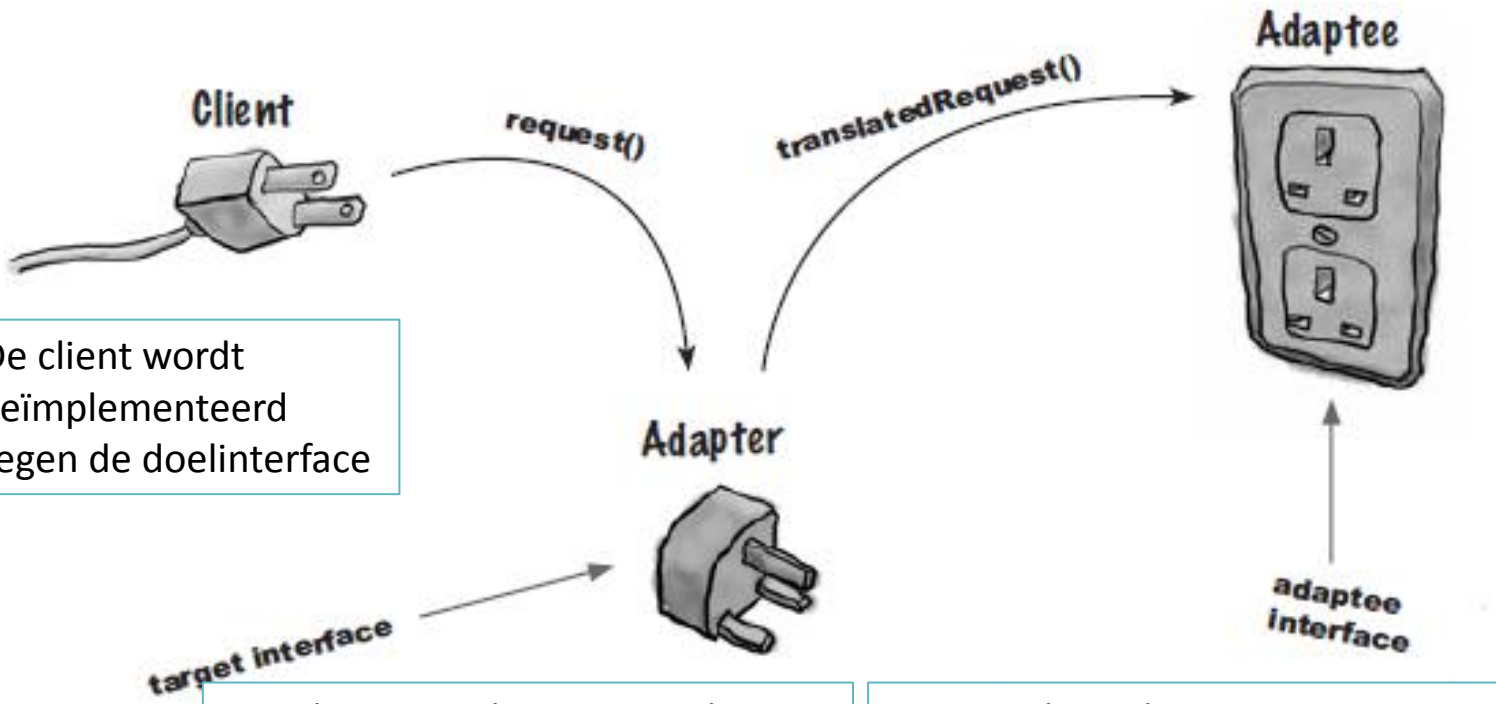
Ook al kennen beide interfaces een methode fly(), kalkoenen vliegen maar korte stukjes, ze kunnen geen grote afstanden afleggen zoals eenden. Om een relatie tussen de methode fly() van Duck en die van Turkey te leggen, roepen we de methode fly() van Turkey vijf maal aan.

# De adapter testen

```
public class Adapter {

   public static void main(String[] args) {
      MallardDuck duck = new MallardDuck();
      testDuck(duck);

      WildTurkey turkey = new WildTurkey();
      Duck turkeyAdapter
            = new TurkeyAdapter(turkey);
      testDuck(turkeyAdapter);
   }

   static void testDuck(Duck duck) {
      duck.quack();
      duck.fly();
   }
}
```

```
run:
Quack
I'm flying
Gobble gobble
I'm flying a short distance
I'm flying a short distance
I'm flying a short distance
I'm flying a short distance
I'm flying a short distance
```

# Adapter pattern




De client wordt geïmplementeerd tegen de doelinterface

De adapter implementeert de doelinterface en bevat een instantie van de adaptee

Stap 1 : client doet een aanvraag aan de adapter door een methode aan te roepen van de doelinterface
Stap 2 : de adapter vertaalt de aanvraag in 1 of meerdere aanroepen naar de adaptee via de adaptee interface
Stap 3 : de client ontvangt de resultaten zonder te weten dat er een adapter bestaat die voor de vertaling zorgt

# Adapter Pattern

Het Adapter Pattern converteert de interface van een klasse naar een andere interface die de client verwacht.
Adapters zorgen ervoor dat klassen samenwerken. Zonder de adapters lukt dit niet vanwege incompatibele interfaces.